Simulation of a large scale dynamic pickup and delivery problem

Esa Hyytiä, Lauri Häme, Aleksi Penttinen and Reijo Sulonen
Helsinki University of Technology TKK, Finland
{firstname.lastname}@tkk.fi

ABSTRACT

We study a variant of dynamic vehicle routing problem with pickups and deliveries where a vehicle is allocated to each service (i.e., trip) request immediately upon the arrival of the request. Solutions to this problem can be characterized as dynamic *policies* that define how each customer is handled by operating a fleet of vehicles. Evaluation of such policies is beyond the grasp of analytical studies and requires extensive simulations. We present an efficient and modular simulation tool developed for studying the performance of a large scale system with different policies under given trip arrival process. Numerical and analytical observations on the model are utilized to provide guidelines for solving the routing problem efficiently, and to support the validation of the simulation results. Application of the developed framework is demonstrated by several numerical examples, e.g., policy parameter optimization, which all give insight on the viability of this type of transportation system.

Categories and Subject Descriptors

G.4 [**Mathematical software**]: *algorithm design and analysis*; I.6.3 [**Simulation and modeling**]: Applications

1. INTRODUCTION

The dynamic vehicle routing problem with pickups and deliveries (VRPPD) involves the dispatching of a fleet of vehicles in real time in order to serve customers requesting transportation of goods or passengers from one location to another. In general, vehicles can serve more than one request at the same time. The main applications of this problem include the transportation of handicapped and elderly people in urban areas (dial-a-ride problem, DARP) and the transportation services of letters and parcels performed by courier companies (urban courier service problem) [4].

In this type of dynamic routing problems, customer requests are revealed gradually in the course of time and thus the vehicle routes are subject to modifications as they are executed. Therefore, solutions to dynamic problems are often characterized as *policies* specifying the actions in different situations. Policies attempt to manage the well-known *dichotomy* between the work the vehicle fleet conducts and the service the passengers obtain, e.g., between the kilometers the vehicles drive and the mean traveling time. Indeed, an efficient policy seeks to optimize either of the two aspects, or some balanced combination of them (cf., Pareto optimality).

In this work, we consider a *large scale dynamic* vehicle routing problem with pickups and deliveries where each customer is assigned to a vehicle immediately at the arrival of the request. In this case the solution to the dynamic VRPPD can be decomposed into two subpolicies: vehicle allocation, and vehicle routing. For each customer the system decides, in on-line fashion, which vehicle takes the customer after which the route of the corresponding vehicle is updated. The vehicle routing part is inherently combinatorial by nature (cf., the traveling salesman problem) and, in many cases, needs to be solved for all vehicles actually *before* a vehicle allocation can be made. Furthermore, as we focus on systems with at least several hundreds of vehicles under high transportation demand, even a simple policy tends to induce extensive computations. Thus, evaluation of the performance of different policies requires an efficient simulation platform tailored particularly for this purpose.

We have developed a modular simulator that is capable of handling various capacity and time constraints in a large system at fast speed. Our goal is two-fold. First, we use the simulator to study this complex transportation system in general in order to understand, e.g., the trade-off between the system's work and the service the passengers experience. Secondly, we apply the simulator to develop efficient policies for solving the problem. In this paper we describe the simulator design, make some fundamental observations on the dynamics of the problem, and demonstrate how the simulator can be used to improve the heuristic vehicle routing and allocation policies by tuning policy parameters.

The rest of the paper is organized as follows. In Section 2, the main concepts of the simulation model are introduced. Vehicle routing and allocation are studied in Section 3. Section 4 describes the simulator design and discusses the most important implementation decisions. In Section 5, a collection of simulation results are given. This is followed by a discussion of the main contributions in Section 6.

1.1 Related work

Most studies related to vehicle routing problems focus on the static case and relatively few results for dynamic problems have been reported. The main solution concepts related to the dynamic VRP, some of which apply to the dynamic

VRPPD, are examined in [20, 11, 8]. Most of the work on the dynamic VRPPD considers a special case of the problem, namely the online dial-a-ride problem (OlDARP), in which no time constraints are present [6]. Only a few analytical results for this problem have been reported.

In [5], it is shown that the competetive ratio[1] of any deterministic algorithm is at least 2 for minimizing route duration and at least $1+\sqrt{2}$ for minimizing the sum of service times. In addition, an algorithm for the first objective with competetive ratio 2 is described for the special case where the vehicle capacity is infinite. In [1], an improved algorithm is presented, for which the competetive ratio is 2 for any capacity value. In [13], a special case of the OlDARP is studied, in which at the release time of a request, only the pickup point is revealed. The authors prove that the competitive ratio for this problem is at least 3 and propose an algorithm with which this lower bound is achieved.

The first exact algorithm for the dynamic single vehicle DARP was introduced in [19]. Because of the inherent complexity of the problem, most of the recent studies related to the dynamic VRPPD have been directed towards heuristic procedures. In [23], lower bounds and constant-factor policies for the uncapacitated multiple vehicle version of the problem are presented. In [21], an algorithm for a real-life multiple vehicle VRPPD with capacity and time constraints is described. A specially tailored objective function for a dynamic environment is used instead of the objective function of the static problem. Motivated by urban courier services, a heuristic for a dynamic uncapacitated VRPPD with time windows is presented in [17]. The authors report that the total distance traveled may be reduced by incorporating a certain waiting strategy. An improved algorithm is introduced in [16]. In [7], a tabu search algorithm for a similar problem is presented and applied to problems with up to 33 requests. In [22], an algorithm for a capacitated dynamic VRPPD is proposed. Computational results indicate that the performance of the algorithm can be improved by up to 22% when information on future requests is taken into account. Finally, in [9], two algorithms for an uncapacitated dynamic VRPPD are described. Computational tests show that by estimating future requests, an improvement of 11-69% can be achieved for both algorithms.

As already stated, this paper focuses on a large scale dynamic VRPPD involving at least several hundreds of vehicles. We introduce a general solution concept particularly tailored for this type of environment and different policies for handling customers with a sub-second interarrival time. The effect of the rate of trip requests and the choice of the solution policy on travel time, distance driven per customer and waiting time are evaluated by means of computational experiments. In addition, several analytical observations related to the level of service and system performance in a dynamic large scale transportation system are presented.

2. MODEL

We consider an abstract model where *trip requests* arrive according to a Poisson process with rate λ [trip/s], and that for each trip request both the pickup and delivery locations are uniformly distributed in a finite convex region with area A (cf. Poisson point process). The trip requests are denoted by a sequence of triples, $(t_i, \mathbf{r}_i^{(1)}, \mathbf{r}_i^{(2)})$, where t_i denotes the time instance (release time) of the ith request, and $\mathbf{r}_i^{(1)}$ and $\mathbf{r}_i^{(2)}$ the origin and the destination (point) of the trip, respectively. We have *a fleet of n vehicles* each with c passenger seats in order to support the given transportation demand in online fashion. We assume Euclidean distances between any two points and thus each vehicle uses *direct path* between the waypoints that define the route. When a trip request arrives, it is immediately assigned to a single vehicle. The chosen vehicle then, at some point of time, picks up the passenger for delivery to the corresponding destination. With $c > 1$ several passengers can share a vehicle in time, which allows one to combine trips and decrease the effort per passenger.

For simplicity, we assume a constant velocity v (e.g., 36 km/h) and a constant (minimum) stop time of t_{st} (e.g., 30 s). After the stop time, passengers can enter and exit the vehicle until the vehicle starts moving again. The stop time is assumed to include deceleration and acceleration of the vehicle. Note that in our trip demand model (Poisson point process), each stop corresponds to exactly one passenger entering or exiting the vehicle at a time, i.e., passengers do not share the stops (cf., door-to-door vs. stop-to-stop service).

We study the performance of such a model when the number of vehicles is large and demand is high with the goal of developing efficient ways to operate the vehicle fleet. Although the model is presented here in the passenger transportation context, the problem itself covers also other applications such as on-demand parcel or message delivery [23].

2.1 Heuristic policies

Let α denote *the policy*, which (i) decides on which vehicle a new customer is allocated to, and (ii) decides on the vehicle's route in dynamic fashion based on the current state of the system. The optimal way to operate a fleet of n vehicles is a very difficult problem, and to start with, requires defining a satisfactory balance between the two conflicting goals: the level of service and the system's efforts.

In this paper, we limit our approach to policies that can be expressed in terms of a cost function $f(\cdot)$ in the following way. A cost function can be evaluated for any given route ξ, which is defined by a sequence of waypoints, i.e., pickup and delivery locations. Also existing state information can be included in the cost function, e.g., in the form of the existing route of a vehicle, denoted by ξ_{old}. Different routes can be evaluated for each vehicle. The new passenger is allocated to the vehicle with the lowest cost route. The particular vehicle also switches to the new route immediately, i.e., the routes are constantly changing in response to the trip requests.

We investigate the performance of the system with three simple but intuitive heuristic policies. Let $t(\xi)$ be the route length (in time) and let $d(\xi)$ be the total remaining travel time of all customers assigned to the vehicle (waiting for pickup or traveling in the vehicle) according to the planned route. For each trip request, the new trip is added to the route of a vehicle with respect to the following criteria:

- `min-RD` (minimum route duration) assigns the new trip to the vehicle which can deliver both the new and its existing passengers fastest:

$$f(\xi, \xi_{\text{old}}) = t(\xi).$$

[1]Online algorithm has a competetive ratio α if its performance for any input is at most α times worse than that of an optimal offline algorithm.

- `min-ΔRD` (minimum difference in route duration) chooses the vehicle (and route) which can serve the new trip request with the smallest additional effort (in time):

$$f(\xi, \xi_{\text{old}}) = t(\xi) - t(\xi_{\text{old}}).$$

- `min-ΔST` (minimum difference in system times) represents the difference between the sum of the passengers' system (sojourn) times before and after inclusion of the new trip request:

$$f(\xi, \xi_{\text{old}}) = d(\xi) - d(\xi_{\text{old}}).$$

Note that global information on the system is utilized only in the allocation phase; the route evaluation can be done locally for each vehicle. Note also that `min-RD` implements a some kind load balancing between the vehicles.

2.2 Performance

2.2.1 Passenger service level

Given a policy, the stochastic system is, in principle, well-defined and one can study its various statistics. In a stable system, each passenger first waits until the assigned vehicle arrives, and then travels in the vehicle to her destination, possibly via an indirect zig-zag route due to the other passengers assigned to the same vehicle. Let the random variable $W_i = W_i(\alpha)$ denote the *waiting time* of customer i, i.e., the length of the time interval from the request until the passenger enters the vehicle. Similarly, let $T_i = T_i(\alpha)$ denote the time customer i spends inside a vehicle (ride time), and $X_i = X_i(\alpha)$ the total distance she travels. Now,

$$\begin{aligned} W_i &> t_{\text{st}} \quad \text{with probability of 1,} \\ T_i &\geq X_i/v + t_{\text{st}}, \\ S_i &= W_i + T_i, \end{aligned}$$

where, $S_i = S_i(\alpha)$ is the sojourn time or *latency* of customer i. We are interested in the mean values, $\mathrm{E}[W]$, $\mathrm{E}[T]$, $\mathrm{E}[S]$, and $\mathrm{E}[X]$ as they describe how efficiently the system works from the passenger point of view. In particular, the mean latency $\mathrm{E}[S]$ is important as it defines the *transportation capability* of the system as observed by the customers. Note that according to the Little's result [14], e.g., the mean number of passengers in the system is $\lambda \cdot \mathrm{E}[S]$.

2.2.2 System performance

In addition to the above metrics, one can study the situation from the vehicles' point of view. In our model, the vehicle is either moving at velocity v, or stopped. Thus, its activity corresponds to an ON/OFF process. Let $B_i^{(j)}$ denote the duration of the ith leg of vehicle j, and $I_i^{(j)}$ the succeeding stopping time, which is at least t_{st}. Assuming policy α treats all vehicles equally, we can basically focus on the mean values $\mathrm{E}[B]$ and $\mathrm{E}[I]$, which provide us all the long term (average) quantities. Firstly, in a stable system the average customer flow in and out are equal:

$$\lambda = \frac{(1/2)n}{\mathrm{E}[B] + \mathrm{E}[I]},$$

where n denotes the number of vehicles and the factor of $1/2$ is due to the fact that only every second stop corresponds to a departing customer. Note that the above assumes that an empty vehicle stays where the last customer is delivered. Substituting $\mathrm{E}[I] \geq t_{\text{st}}$ and $\mathrm{E}[B] \geq 0$ in the above yields a *capacity constraint*,

$$\lambda_{\max} \leq \frac{n}{2\,t_{\text{st}}}. \tag{1}$$

Similarly, the *mean number of moving vehicles* is given by

$$\mathrm{E}[\text{moving}] = \frac{n\,\mathrm{E}[B]}{\mathrm{E}[B] + \mathrm{E}[I]}. \tag{2}$$

A key performance metric for the system is the *mean distance driven per passenger* and denoted by τ. The average collective velocity of the fleet is $n\mathrm{E}[B]/(\mathrm{E}[B]+\mathrm{E}[I])\cdot v$ [m/s]. Passengers rate was λ [1/s], and thus

$$\tau = \frac{n\,v\,\mathrm{E}[B]}{\lambda\,(\mathrm{E}[B] + \mathrm{E}[I])}, \tag{3}$$

i.e., the amount of work (in metres) the fleet conducts in order to fulfil a single trip on average. Note that the τ is constrained not only by the speed of the vehicles but also by the time the vehicle is stopped. A useful way of characterizing the "resource" of the system is to consider the *effective speed* of a vehicle. Assume that the vehicle is under such a demand that it never stays stopped longer than the minimum time for a passenger to enter or exit the vehicle. Each passenger requires a pickup and delivery resulting in a delay of $2t_{\text{st}}$. Given that each vehicle serves on average λ/n requests per unit time, we have for the effective speed,

$$v_{\text{eff}} \leq v\left(1 - \frac{\lambda 2 t_{\text{st}}}{n}\right).$$

Consequently, the distance driven per passenger is constrained by the *effective speed bound*:

$$\tau = \frac{n\,v_{\text{eff}}}{\lambda} \leq \left(\frac{n}{\lambda} - 2\,t_{\text{st}}\right) v. \tag{4}$$

3. ROUTING AND ALLOCATION

Recall that when a trip is ordered from the system it is immediately allocated a vehicle that will handle the trip. This decomposes the policy into two subpolicies, vehicle allocation and routing. In this paper we limit ourselves to policies where allocation and routing decisions are based on cost functions: When a trip request arrives all vehicles compute a candidate route based on a cost function and then the vehicle with the lowest cost route is selected.

In this case the allocation is simple and scales linearly to the number of vehicles, but computing the candidate route is somewhat more demanding. The problem is inherently combinatorial in nature; when a new trip request arrives the vehicle needs to combine the new pickup and delivery with the previously routed requests, i.e., with the existing route. In principle, this requires enumeration of possible routes and evaluating the cost function in each one of them. We assume that the route of a vehicle is simply defined as the order of (pickup and delivery) waypoints that the vehicle passes by. Possible routes are limited only by the fact that a pickup must take place before the corresponding delivery.

In this section we describe two experiments that aim to shed light on two important questions that arise from our approach. First, by locking the vehicle and the trip already at the arrival of the request we achieve several benefits; (i) decomposition - the vehicles can independently solve their routing problem without consulting with other vehicles, (ii)

the solution space becomes significantly smaller thus facilitating the computational burden, and (iii) customer can be immediately notified the identity of the vehicle that handles the request. However, these benefits come with a performance cost. This question will be elaborated in Section 3.1.

In the literature, this kind of routing problems are often solved by *insertion heuristics*, in which a new trip is added in such a way that the relative order of the existing waypoints is preserved. Although such an approach enables a polynomial running time, the associated performance loss seems inevitable. Somewhat surprisingly, for large systems the performance loss appears to be neglible, as will be motivated in Section 3.2 and evaluated by simulations in Section 5.4.

3.1 Immediate allocation

A routing and allocation policy, in which a vehicle is fixed for each customer at the release time of the request, will not generally perform as well as a policy in which customers may be exchanged between vehicles at any instant. This is due to the fact that the appearance of a new customer may render some of the existing customer-vehicle assignments suboptimal. In the following examination, we will estimate the performance decrement due to immediate allocation policy.

For simplicity, let us study a static vehicle routing problem involving two vehicles, in which a single service point chosen randomly from the unit square is associated to each customer. The goal is to assign the customers to the two vehicles and generate routes for the vehicles in a way that the total route length is minimized. We will compare the difference between (i) the optimal solution, in which all possible partitionings of customers among the two vehicles are considered and (ii) a solution based on sequential allocation, in which the customers are assigned to the vehicles one by one. In the second method used to model the immediate allocation policy, the vehicle for which the increase in the route length is minimized, is selected for each customer. Comparing the two solutions gives us an idea on how the immediate allocation method might perform in a dynamic setting, compared to a solution method in which customers may be exchanged between vehicles at any instant.

The relative increase in the total route length of the two vehicles obtained by the sequential allocation method compared to the exact solution is shown in Figure 1. The solid line represents the sequential allocation method in which the customers are assigned in a random order. The relative deviations were computed by means of the formula $\sum_k r_k / \sum_k e_k - 1$, where $\sum_k r_k$ and $\sum_k e_k$ denote the sums of total route lengths acquired by the random order allocation and exact offline algorithms in 500 runs. The dotted and dashed lines represent the corresponding relative deviations obtained by choosing the best and worst assignment orders (out of 100 randomized orders for each run) of customers.

The curves in the figure indicate that the difference between the exact solution and the sequential allocation method increases with the number of customers. The worst ordering produces an increase of approximately 10% on the total route length for the problem involving seven customers. On the other hand, with the best ordering of customers, the increase compared to the static solution is less than 0.5% for each of the studied problems. Generally, it can be stated that the order in which the customers are assigned to vehicles has a substantial effect on the performance of the sequential allocation method. However, even if the customers

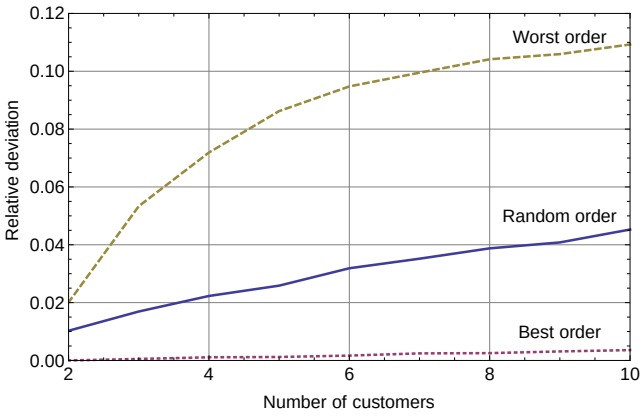

Figure 1: Difference between the solutions to a vehicle routing problem with two vehicles obtained by the exact and sequential allocation methods. The solid line represents the mean relative increase in route length, when the customers are assigned to vehicles in a random order, over 500 runs. The dotted and dashed lines represent the corresponding increase produced by the best and worst assignment orders of customers (out of 100 orders for each run).

were assigned in the worst possible order, the increase compared to the optimal solution is limited to relatively small values.

In this paper we focus on modeling services in which each customer is given an instant response and the allocation has to be executed immediately upon a request. Thus, the allocation order is defined by the arrival process and cannot be optimized. Nevertheless, the above results suggest that by freely exchanging customers among vehicles, only a relatively small improvement in performance can be achieved. In other words, immediate allocation can be considered as a relatively efficient method for solving dynamic problems.

3.2 Insertion vs. enumeration

Clearly, in addition to the allocation order, the quality of the solutions produced by the immediate allocation procedure is strongly dependent on the algorithm used to solve the route for each vehicle. A complete enumeration algorithm, as used in the previous experiment, will in general produce the best possible results with respect to a given routing and vehicle selection policy. In some situations, however, the use of such an algorithm may not be feasible since it would require too much computational work. In the following experiment, the difference between the solutions produced by an exact single vehicle algorithm and the intuitive insertion algorithm is evaluated.

More specifically, we will study the effect of problem size on the difference between the performance of the insertion algorithm and the exact algorithm. The algorithms are tested on a static vehicle routing problem involving 1 to 5 vehicles and 1 to 10 customers per vehicle. Similarly as in the previous experiment, the immediate allocation policy is used. The total route length obtained by the two algorithms with respect to the number of customers per vehicle is shown in Figure 2. Table 1 shows the relative increase in the total route length acquired by using the insertion heuristic with

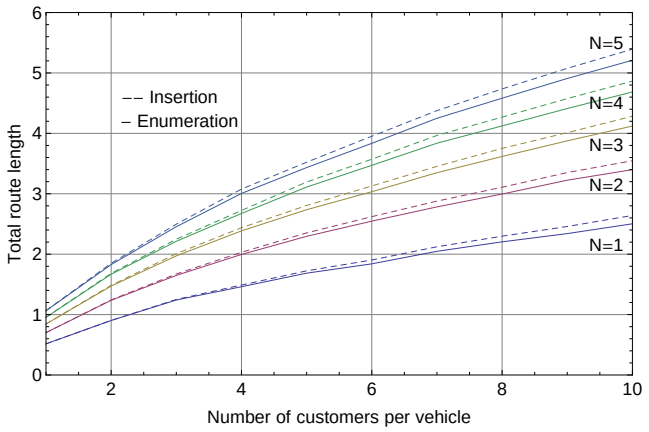

Figure 2: Difference between the total route length obtained by the enumeration and insertion algorithms. The difference between the algorithms increases with the number of customers per vehicle.

customers per vehicle = 10					
number of vehicles	1	2	3	4	5
increase (%)	5.5	4.3	3.9	3.7	3.5

Table 1: Relative increase in the total route length obtained by comparing the insertion to the enumeration with 10 customers per vehicle. Performance loss decreases as the number of vehicles increases.

10 customers per vehicle.

In general, the results indicate that the performance of the insertion heuristic decreases as the number of customers per vehicle increases. Since the curves representing the insertion and enumeration algorithms converge when the problem size is decreased, the use of the classical insertion heuristic can be seen to be well motivated in problems in which the number of customers assigned to a single vehicle is sufficiently small at any instant. This observation is in line with the fact that the insertion algorithm is asymptotically optimal. Indeed, when the number of customers assigned to a single vehicle is limited to 2 or less, the insertion algorithm goes through all possible permutations (even if both a pickup and a delivery node were associated to each customer).

Furthermore, by looking at Table 1, it can be seen that the gap between the two algorithms is slightly decreased as the number of vehicles is increased. Thus, it may be suggested that the insertion algorithm will perform well in problems in which the number of vehicles is large. We will return to this question with simulation results in Section 5.4.

4. SIMULATOR DESIGN

In this section we will briefly describe the design of our simulator tool and some important implementation decisions that provide us reasonably fast simulation times. As mentioned, our goal is to study a system with a large number of vehicles n (e.g., $n = 500$) and a high demand of trips. This means a relatively complicated system with a huge number of state information and internal constraints that must be checked constantly throughout the simulation.

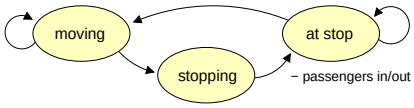

Figure 3: State of each vehicle follows the depicted pattern. The "stopping" state is deterministic delay that models braking, acceleration and delay due to the boarding and alighting passengers. Self-transitions correspond to plan changes.

Thus, even if the policy deciding on vehicle assignment and routes was computationally lightweight, simulating the system with constant parameters for a sufficiently long time period can easily take infeasible amount of real time if the simulator is not implemented efficiently.

However, the policies are hardly computationally light. The core decision each vehicle must do is to decide on route if a new trip were assigned to it. Unfortunately, the number of feasible routes (at each point in time) can be very large. For example, if a vehicle is full and has, say, $c = 10$ passengers and they can be dropped in $10! = 3628800$ different orders. Adding the new trip and the passengers already waiting for a pickup (and delivery) increases the complexity of the problem even further. Considering that each vehicle must evaluate the routes for all arriving trip requests, even simulating performance of one policy at a single load level can be an overwhelming task. Some policies may also contain parameters to be optimized, which sets even higher requirements for the simulation speed.

In order to have a maximal simulation speed, we decided to implement our simulator from scratch using standard C. This approach has also other benefits: (i) The simulator is relatively compact as there are no unnecessary features. (ii) The architecture can be tailored to match our objectives. (iii) Standard C implementation makes the simulator highly portable (we are using it in both Linux and Windows systems). On the downside, we had to re-implement some standard components available in almost any simulation library such as pseudo random number generation and event based scheduling. This, however, is a straightforward task, e.g., by following the steps laid out in [15, 2, 12]. Next we will describe the essential features of our simulator architecture.

4.1 State Description

4.1.1 *Passengers*

The life of a passenger in our system is very similar to jobs in a queueing system. Upon arrival a passenger is either (i) *accepted* and assigned to some vehicle, or (ii) *rejected*. The accepted customers first (iii) *wait for the pickup*. Next the customer (iv) *enters the vehicle* and the system starts to actually process her transportation need with the difference that here it is also possible to conduct "negative work" by moving the passenger further away from her destination. Finally, at some point in time, the vehicle reaches the destination and passenger (v) *exits the system*.

4.1.2 *Vehicles*

The number of vehicles in our model is assumed to be a constant n, and each vehicle is in one of the following three states: (i) *At stop* waiting for (more) passengers, (ii) *Moving* towards the next waypoint, and (iii) *Stopping* phase after a

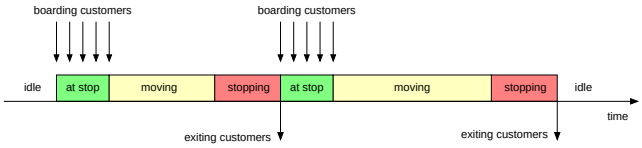

Figure 4: Realization of vehicle's state as a function of time. The duration of "stopping" state is a constant, while "at stop" state can have a zero duration.

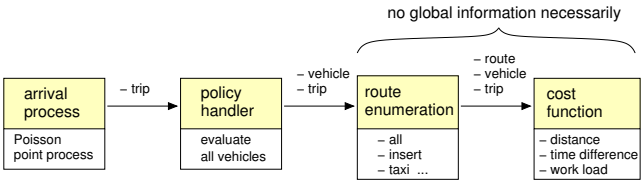

Figure 5: Modular decision chain from the request generation to evaluation of the cost function.

transition.

This is illustrated in Figures 3-4. Note that the state "stopping" includes braking, acceleration and other unavoidable delays per stop. The state "at stop" means the vehicle is free to go at any moment. Also the bookkeeping of boarding and alighting passengers occurs in this state. In particular, the duration of the stoppping state can be zero, except when the vehicle extends the stay for some reason, e.g., when it is idle. In addition to this, each vehicle maintains an ordered list of waypoints (worklist), which defines its current route. This list can be modified in response to each new customer with exception of the first waypoint in case the vehicle has started the corresponding stopping state. This is illustrated with the self-transitions in Figure 3.

4.2 Policy Architecture

As discussed previously, we define policy α by means of cost functions. That is, each vehicle is basically asked (to estimate) how much it costs if a new trip request is assigned to it. In order to estimate the vehicle specific cost, one generally needs to consider some set of possible routes. To this end, the simulator provides an unified interface which separates the route enumeration and the cost function evaluation. Such a cost function f then defines policy α if we always choose the vehicle and the route which yields the lowest cost at that point in time. This decision chain is illustrated in Figure 5. We observe a modular design, where each block can be replaced without modifying the others.

4.2.1 *Arrival process*

Arrival process module is responsible for generating the trip requests. In this paper, we assume a Poisson point process in a circular area, while the simulator design allows any other more specific arrival process and area.

4.2.2 *Policy handler*

This module collects the information (cost estimates) from the vehicles and then assigns the new trip to the most suitable vehicle. That is, the *vehicle allocation* occurs at this point. Note that the example policies considered in this paper are all such that the global information is only available at this point. That is, each vehicle independently evaluates the cost incurred if the given trip is assigned to it. However, the modular design of the simulator allows that also the vehicles can take into account global information if seen relevant in the particular case.

4.2.3 *Route enumeration*

The simulator provides several options to define the subset of all routes to be considered upon a new trip request:

1. *taxi*: New trip is inserted in such a way that no two trips share the vehicle at the same time.
2. *insertion*: New trip is inserted in such a way that the relative order of the existing waypoints is unchanged.
3. *all*: Consider all orders of the waypoints, exhaustive and thus the number of potential routes may be huge.

Clearly, *taxi* $\subset$ *insertion* $\subset$ *all*. The *taxi* subset limits the performance severely. However, in large scale system with a large number of vehicles, as we discussed previously and will also show in numerical experiments, the *insertion* and *all* provide a similar performance level.

With *insertion* and *all*, it is important to prune infeasible routes efficiently. To this end, we have two types of constraints: (i) *time constraints* that are typically deadlines for arriving to each waypoint, and (ii) *capacity and order constraints*, i.e., a vehicle may not pickup more passengers than its capacity allows, and pickup must be before the corresponding delivery. The time constraints can be given either "external", e.g., one can require that each customer must be picked up within 10 minutes from the request, or they can be based on the currently known best route when evaluating the routes (policy specific).

Assume that while enumerating the routes at some stage we have fixed the first k waypoints and the task is to choose the next. If some remaining waypoint x cannot be reached in time anymore, then all routes with this initial sequence are infeasible and can be excluded. The same does not hold for capacity or order constraint. Unlike time, the occupation can still (and will) decrease. Similarly, the pickup can be scheduled before the corresponding delivery. In summary, multiple ways to prune the set of routes efficiently exist, of which the (potential) time constraints are more "definitive" due to the nature of time.

4.2.4 *Cost function*

This function returns some real number corresponding to the relative cost of the given routing decision. As discussed previously, with the `min-RD` policy the idea was to minimize the planning horizon. Thus, the time instance $t(\xi)$ representing the time when given vehicle becomes empty is returned when the cost function is called. Note that the modular design enables fast prototyping of new policies without a need to re-implement, e.g., the route enumeration repeatedly.

4.3 Running the simulator

For each simulation run the user must specify (i) the area and passenger arrival rate, (ii) the number of vehicles, their capacity, velocity and the stopping time, and (iii) the fleet operating policy. Additionally one must also decide on the simulation and warm-up periods. All these parameters can be conveniently tuned from a command line interface.

By default the simulator outputs a summary report, which includes all the main performance quantities (essentially to-

trip request rate:	2/s
area:	disk with 5km radius
vehicles:	500, each with 10 seats
velocity:	10 m/s
stopping time:	30 s

Table 2: Basic simulation parameters.

tals and mean values). However, when necessary, one can also enable various log-files to which more detailed information during the simulation run is written. The most important are perhaps the vehicle log file (actions taken by the vehicles) and trip log file (per trip information). Based on these log-files one can obtain, e.g., waiting and travelling time distributions, or study how the direct trip length affects the realized trip length and the number of additional stops. This type of information is vital when one is, e.g., developing better policies.

4.3.1 *Validation*

Before proceeding further with performance evaluation, it is important to ensure that the simulator works correctly. Simulation results to this end are in good agreement, e.g., with the following analytical observations, which support the validity of our implementation:

- Mean trip request length can be computed analytically (cf., random waypoint mobility model [3, 18, 10]), which for Table 2 disk area gives about 4527m. This is used to assess the module generating the trip request.
- Simulating a single vehicle when $\lambda \to 0$ converges to a system where each arriving customer observes the single vehicle idle. Thus, the work the single vehicle does, assuming a work conservative policy, is two times higher than the direct trips.
- Similarly, with a low demand, a huge number of vehicles, and a policy that assigns the trip to the nearest (idle) vehicle yields a system where the ratio of vehicle kilometres to passenger kilometres approaches one.
- With a high demand and efficient policies the driven kilometers per passenger approach the bound (4).

4.3.2 *Simulation speed*

Simulation speed turned out to be more than satisfactory for our purposes. For example, it takes only about 5 minutes of real time to simulate a 10 hour time interval with Table 2 parameters using a standard PC. This already corresponds to a rather high load as an average 72000 passengers are processed. Further speed improvements can be obtained by code optimization and multi-threading. Indeed, both our problem and the design of the simulator lend themselves well to parallel computation. For example, the vehicle specific relative costs can be computed in parallel.

5. NUMERICAL ANALYSES

In this section we demonstrate the simulator and investigate the performance of the defined policies. We use the simulation parameters defined in Table 2, unless otherwise mentioned. We omit the specific results of the `min-ΔRD` policy as it turns out to perform particularly badly in this case resulting in mean travel times of several hours, but use it instead as a component of a parameterized policy.

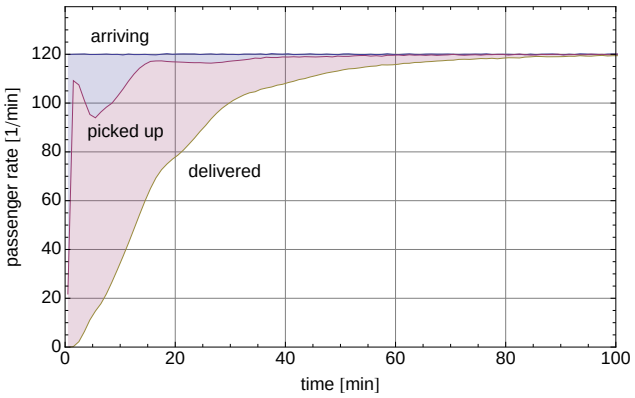

Figure 6: Mean rate of arriving, picked up and delivered passengers over 10000 simulation runs starting from an empty system. The initial transient period is order of 2 hours.

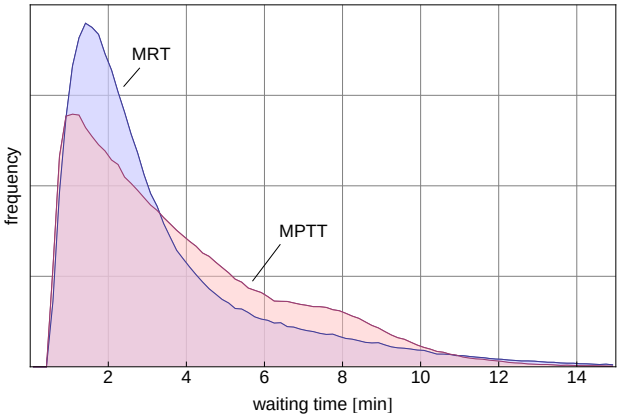

Figure 7: Waiting time distribution for `min-RD` and `min-ΔST` policies with Table 2 parameters. Most passengers wait only few minutes before the pickup.

5.1 Warm-up period

When the simulation is started from a state where all vehicles are empty, the initial state has a strong but transient effect on the behavior of the system. In order to get some idea about the length of the initial transient we next ran 10000 experiments using the `min-RD` policy. The result is shown in Figure 6, where the initially highest curve corresponds to the *passenger arriving rate*, the middle curve to the *passenger pickup rate*, and the lowest to the *passenger delivery rate*. The pickup rate catches the arrival rate relatively fast, while it takes a somewhat longer time before the system's output rate (alighting passengers) stabilizes. In particular, we observe that at least a 2 hour warm-up period should be used in this case. In order to be on a safe side, in the following experiments we use 10 hour warm-up period.[2]

[2]Even though in real-life the daily 24 hour rhythm implies that the trip demand will not be constant for 10 hours.

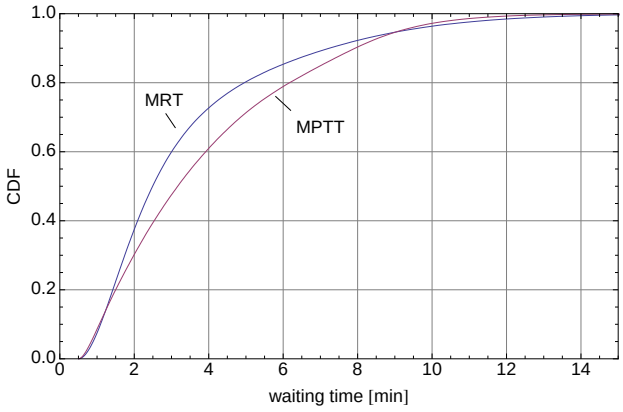

Figure 8: CDF of the empirical waiting time distribution for `min-RD` and `min-ΔST` policies with Table 2 parameters.

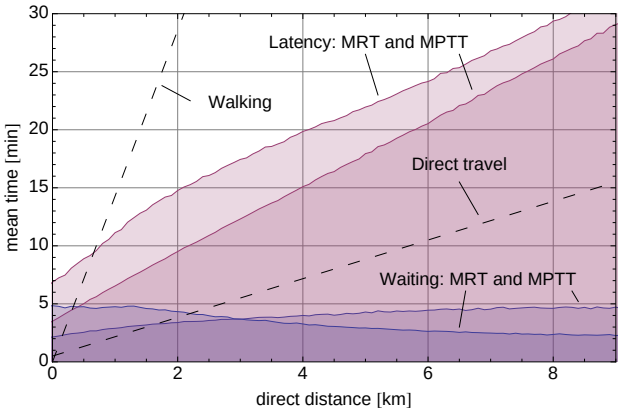

Figure 9: Empirical mean waiting time and latency (sojourn time) with `min-RD` and `min-ΔST` policies conditioned on the direct distance of the trip. Walking time with 70m/min, and the direct driving time with a private car are illustrated with dashed lines.

policy	mean	tail $\mathrm{P}\{W > t\}$		
	$\mathrm{E}[W]$	5min	10min	15min
`min-RD`	3min 25sec	19.8%	3.8%	0.4%
`min-ΔST`	3min 54sec	28.8%	2.9%	0.1%

Table 3: Tail probabilities of the waiting time.

Moreover, note that the area between the passenger arriving rate and the passenger pickup rate curves corresponds to the mean number of waiting customers in the equilibrium, $\mathrm{E}[N_{\mathrm{w}}]$. Similarly, the area between the pickup and delivery rates is equal to the mean number of customers in the vehicles, $\mathrm{E}[N_{\mathrm{v}}]$. Thus,

$$\mathrm{E}[N_{\mathrm{w}}] = \int_0^\infty \lambda - p(t)\,dt, \text{ and } \mathrm{E}[N_{\mathrm{v}}] = \int_0^\infty p(t) - d(t)\,dt,$$

where $p(t)$ denotes the expected pickup rate at time t, and $d(t)$ the mean delivery rate at time t. Obviously, $\mathrm{E}[N_{\mathrm{v}}]$ is bounded from the above by the total capacity of the fleet.

5.2 Waiting time distribution

The simulator provides various log-files, from which one can extract more detailed statistics when necessary. Figures 7-8 illustrate the empirical waiting time distribution as observed by the passengers when the fleet is operated according to the `min-RD` and `min-ΔST` policies. We observe that the `min-RD` policy picks up the passengers somewhat quicker than the `min-ΔST`. Passenger arrival rate $\lambda = 2/\mathrm{s}$ is already a rather high demand for the given $n = 500$ vehicles to handle (see Section 5.4). Despite of this, the waiting times are actually reasonable and most passengers are picked up with 5 minutes, and only very few have to wait more than 15 minutes, as shown in Table 3.

5.3 Latency vs. trip distance

Waiting time does not necessary depend on the trip distance. In contrast, the latency, i.e., the time from the request until the delivery in general correlates strongly with the trip distance due to the finite velocity of the vehicles. Figure 9 illustrates how the mean waiting time and latency depend on the trip distance. The x-axis corresponds to the direct distance in kilometres. On the y-axis we have both the waiting time and the latency in minutes. The dashed line depicts the latency assuming private cars. The `min-ΔST` policy clearly outperforms the MRT. We also note that `min-RD` gives a higher priority to long trips as such trips tend to extend the planning horizon further than the short trips. This, however, may not always be an optimal strategy, and indeed, with `min-ΔST` the situation appears to be more fair.

5.4 Insertion vs. enumeration

In Section 3.2 we motivated that in a large system with many vehicles there may not be need to enumerate all routes but a simple insertion heuristic would perform nearly as well. Figure 10 represents the mean travel time as a function of load, computed for both the insertion heuristic (dashed curves) and the full enumeration (markers) with `min-RD` and `min-ΔST` policies. As expected, in this large system with 500 vehicles, the difference between the insertion and enumeration methods is almost negligible. Looking at the distance driven per passenger metric we observe the same result. We omit the illustration for brevity.

5.5 Policy optimization

Viability of this kind of transportation system depends on the performance of the applied policies. Simulations are needed for experimenting with different policies. In this example we evaluate the performance of the system with `min-RD` and `min-ΔST`. In addition, we consider a parameterized policy, where the cost function for a trip request is defined as

$$f(\xi,\, \xi_{\mathrm{old}},\, p) = p \cdot \big(t(\xi) - t(\xi_{\mathrm{old}})\big) + (1-p)\big(d(\xi) - d(\xi_{\mathrm{old}})\big),$$

where the policy parameter p defines the weighting between the two objectives (i.e., the policy has $100p\%$ of `min-ΔRD` and $100(1-p)\%$ of `min-ΔST`) and is subject to optimization. Clearly, this policy represents a balance between the driven distance and passenger travel time; `min-ΔRD` attempts to add the new trip request so that the additional work is as small as possible, whereas `min-ΔST` greedily minimizes the sum of all travel times in the system. Note that `min-ΔRD` by itself causes excessive delays in this setting (of the order

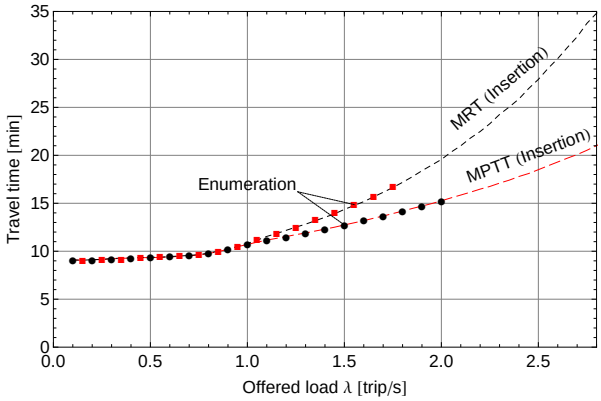

Figure 10: Travel time as a function of load. Insertion heuristic (lines) provide a scalable way of solving the routing subproblem without almost any difference in performance compared to the full enumeration (markers). Full enumeration becomes computationally infeasible at large load values.

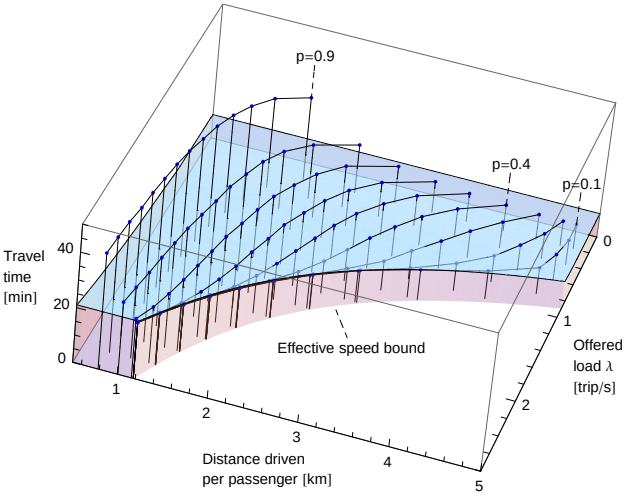

Figure 11: Performance of the system with different values of policy parameter p (cf. bound (4)).

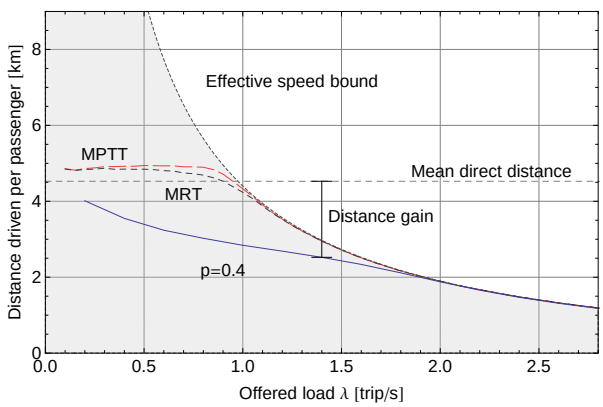

Figure 12: Driven distance per passenger as a function of offered load. The mean length of requests is 4.527km, which equals to the driven kilometers per passenger if private vehicles. Shaded area corresponds to the feasible region defined by (4).

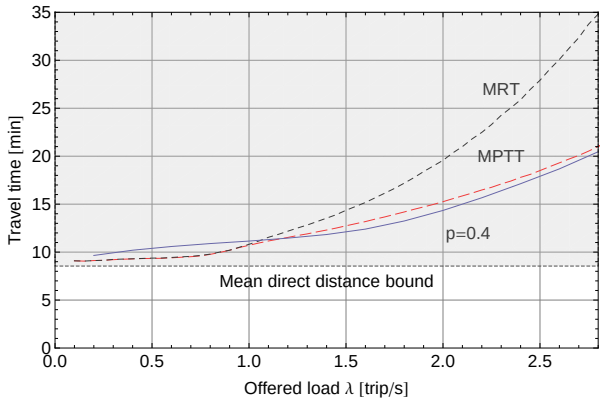

Figure 13: Mean travel time. Mean direct distance bound corresponds to the travel time in a system where all trips are driven in private vehicles.

of several hours) and is not considered here. In optimizing a policy parameter an efficient simulator is invaluable as each parameter adds a new dimension to the parameter space.

Figure 11 shows the mean travel time and distance driven per passenger as a function of load for different values of the policy parameter p. We observe that even a small increase in the mean travel time can be efficiently converted into saved distance and in some cases also the travel time decreases even below the `min-ΔST` profile (shown as an opaque box in the figure). Including an `min-ΔRD` component into the policy allows us to take into account the additional work induced by the new trip request. By allocating the trip to a vehicle that can handle the trip with little additional time the overall performance can be improved.

To show the results in more traditional form, let us select the policy with the parameter value $p = 0.4$ and study its performance in detail. Figure 12 shows the distance driven per passenger as a function of load and compares it to that of `min-ΔST` and `min-ΔRD`. Whereas these comparison heuristics tend to start moving the vehicles even at low loads our parameterized policy $p = 0.4$ performs clearly more efficiently in this respect. The difference between mean direct distance and distance driven per passenger is coined as *the distance gain*, cf. the figure, and reflects the amount of kilometers saved per passenger by using this transportation system instead of serving the trip requests by private vehicles. All policies utilize the available kilometers quite quickly and afterwards follow the effective speed bound (4) very closely.

Figure 13 depicts the travel time (including waiting time) profiles of the compared policies. It can be seen that as soon as the distance approaches its upper bound the delays start to increase. `min-ΔST` shows rather moderate increase in the travel time, but `min-RD` performs significantly worse. The parameterized heuristic $p = 0.4$ is reasonably good especially at higher loads, which makes it a good compromise between the two conflicting objectives.

6. CONCLUSIONS

In this paper we have considered a dynamic behicle routing problem with pickups and deliveries. Our focus was on systems where a large number of vehicles are needed to support the transportation demand. As a particular feature of our system, a vehicle is assigned to each passenger immediately upon the trip request. In this context, we have described a specifically tailored simulator framework, that can be used to evaluate the transportation system and to rapidly prototype new operating policies for the vehicle fleet. The main objectives have been efficiency and modularity.

One of the most important design choices for a vehicle routing policy is the set of routes considered for each trip request. We have shown, both by means of analysis and simulation experiments, that in our context it is typically sufficient, without any significant loss in performance, to consider the insertion approach for route enumeration, where the relative order of the earlier waypoints is always kept the same. That is, it is not necessary to enumerate *all* the feasible orders of waypoints per trip request and per vehicle, which indeed can take some time in a large system.

On the other hand, we have also demonstrated the viability of this type of transportation system. In general, there is a well-known trade-off between the work conducted (driven kilometers) and the level of the service (e.g., mean waiting times). However, our experiments suggest that if the passengers are willing to accept even a small average delay for their trips, in form of waiting time and/or a longer route, then the amount of work can be reduced considerably. That is, the transportation cost per trip can be reduced significantly.

7. ACKNOWLEDGMENTS

This work was conducted in Metropol project that is supported by the Finnish Funding Agency for Technology and Innovation, Finnish Ministry of Transport and Communications, Helsinki Metropolitan Area Council and Helsinki City Transport. The authors would like to thank Dr. Samuli Aalto, Mr. Teemu Sihvola and Prof. Jorma Virtamo for their invaluable comments while preparing this paper.

8. REFERENCES

[1] N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *STACS 2000 Lecture Notes in Computer Science*, volume 1770, pages 639–650. Springer, Berlin, 2000.

[2] J. Banks, J. S. C. II, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall International Series in Industrial and Systems Engineering. Prentice-Hall, third edition, 2001.

[3] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Trans. on Mobile Computing*, 2(3):257–269, Jul.–Sept. 2003.

[4] J.-F. Cordeau, G. Laporte, J.-Y. Potvin, and M. Savelsbergh. Transportation on demand. In *Transportation*, pages 429–466. Amsterdam: North-Holland, 2007a.

[5] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268:91–105, 2001.

[6] G.Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 2009. In press.

[7] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C*, 14:157–174, 2006.

[8] G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151:1–11, 2003.

[9] G. Ghiani, E. Manni, A. Quaranta, and C. Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E*, 45:96–106, 2009.

[10] E. Hyytiä, P. Lassila, and J. Virtamo. Spatial node distribution of the random waypoint mobility model with applications. *IEEE Trans. on Mobile Computing*, 5(6):680–694, June 2006.

[11] A. Larsen. The dynamic vehicle routing problem. *PhD thesis, Technical University of Denmark*, 2000.

[12] P. L'Ecuyer. Random number generation. In *Handbook of Computational Statistics*, chapter 2, pages 35–70. Springer-Verlag, 2004.

[13] M. Lipmann, X. Lu, W. E. de Paepe, R. A. Sitters, and L. Stougie. On-line dial-a-ride problems under restricted information model. *Algorithmica*, 40:319–329, 2004.

[14] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operations Research*, (9):383/387, 1961.

[15] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1), Jan. 1998.

[16] S. Mitrovic-Minic, R. Krishnamurti, and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B*, 38:669–685, 2004.

[17] S. Mitrovic-Minic and G. Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B*, 38:635–655, 2004.

[18] W. Navidi and T. Camp. Stationary distributions for the random waypoint mobility model. *IEEE Trans. on Mobile Computing*, 3(1):99–108, Jan-Mar 2004.

[19] H. Psaraftis. A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14:130–154, 1980.

[20] H. Psaraftis. Dynamic vehicle routing: status and prospects. *Annals of Operations Research*, 61, 1995.

[21] M. Savelsbergh and M. Sol. DRIVE: Dynamic routing of independent vehicles. *Operations Research*, 46, 1998.

[22] D. Sáez, C. Cortés, and A. Núñez. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. *Computers and Operations Research*, 35:3412–3438, 2008.

[23] H. Waisanen, D. Shah, and M. Dahleh. A dynamic pickup and delivery problem in mobile networks under information constraints. *IEEE Trans. on Automatic Control*, 53:1419–143, 2008.